Improving Context-Awareness in Self-Adaptation using the DYNAMICO Reference Model

Gabriel Tamura*, Norha M. Villegas*†, Hausi A. Müller†, Laurence Duchien‡, Lionel Seinturier‡
*Department of Information and Communications Technologies, Icesi University, Cali, Colombia
gtamura@icesi.edu.co
†Department of Computer Science, University of Victoria, Victoria, Canada
{nvillega, hausi}@cs.uvic.ca
‡INRIA Lille-Nord Europe - LIFL - University of Lille 1, Lille, France
{laurence.duchien, lionel.seinturier}@inria.fr

***Abstract*—Self-adaptation mechanisms modify target systems dynamically to address adaptation goals, which may evolve continuously due to changes in system requirements. These changes affect values and thresholds of observed context variables and monitoring logic, or imply the addition and/or deletion of context variables, thus compromising self-adaptivity effectiveness under static monitoring infrastructures. Nevertheless, self-adaptation approaches often focus on adapting target systems only rather than monitoring infrastructures. Previously, we proposed DYNAMICO, a reference model for self-adaptive systems where adaptation goals and monitoring requirements change dynamically. This paper presents an implementation of DYNAMICO comprising our SMARTERCONTEXT monitoring infrastructure and QOS-CARE adaptation framework in a self-adaptation solution that maintains its context-awareness relevance. To evaluate our reference model we use self-adaptive system properties and the Znn.com exemplar to compare the Rainbow system with our DYNAMICO implementation. The results of the evaluation demonstrate the applicability, feasibility, and effectiveness of DYNAMICO, especially for self-adaptive systems with context-awareness requirements.**

I. INTRODUCTION

Self-adaptation mechanisms, driven by adaptation goals (i.e., functional and non-functional requirements), monitor entities from the target system environment to decide when and how to perform adaptations [1]. Nevertheless, when adaptation goals change while the system executes, both monitoring and self-adaptation mechanisms may become inapplicable thus compromising the accomplishment of the target system objectives and the assurance of its quality properties [2]. This happens because either the adaptation mechanism addresses out-of-date goals, or the monitoring mechanism addresses monitoring requirements that are irrelevant to the actual adaptation goals.

In a previous research work we proposed DYNAMICO (Dynamic Adaptation, Monitoring and Control Objectives model), a reference model that provides guidelines for designing and implementing self-adaptive software (SAS) systems where both adaptation and monitoring infrastructures require self-adaptive capabilities [3]. That is, DYNAMICO targets specially SAS systems that must cope with changes in adaptation goals and context monitoring requirements at runtime. DYNAMICO specifies three types of feedback loops: (i) the control objectives feedback loop, which keeps track of changes in adaptation goals; (ii) the target system adaptation feedback loop, which models the target system adaptation mechanism; and (iii) the dynamic monitoring feedback loop, which models a self-adaptive monitoring mechanism. Adaptation goals, controlled by the first feedback loop, govern the dynamic behavior of the other two feedback loops. Besides raising the visibility of feedback loops—an important concern in the engineering of SAS systems [4], [5], DYNAMICO characterizes the separation of concerns and interactions among the different types of feedback loops required to engineer this kind of systems.

Our contribution in this paper is twofold. First, we realize an implementation of the DYNAMICO reference model that demonstrates its applicability to realize SAS systems enabled with dynamic monitoring infrastructures. The purpose of a dynamic monitoring infrastructure is to maintain the relevance of the SAS deployed monitoring elements with respect to changing adaptation goals. Our implementation integrates the SMARTERCONTEXT monitoring infrastructure [6] and the QOS-CARE adaptation framework [7], contributions of our previous research. SMARTERCONTEXT realizes the dynamic monitoring feedback loop specified by DYNAMICO. QOS-CARE realizes the adaptation framework to reconfigure both the dynamic monitoring infrastructure of SMARTERCONTEXT and the target system. Second, we present a comparative evaluation of our implementation against Rainbow/Znn.com [8]. We selected Rainbow because it has been recognized by the *software engineering for adaptive and self-managing systems (SEAMS)* research community as a reference approach to architectural self-adaptation, and the availability of quantitative data about its evaluation. We conducted an experimental case study to analyze the feasibility and effectiveness of DYNAMICO to improve context-awareness of SAS solutions derived from it. Our case study focuses on a dynamic SOA governance scenario where Znn.com is used as a service-oriented target system to be adapted in order to guarantee service level agreements (SLA), which are renegotiated at runtime. Given that Znn.com was explicitly developed to evaluate the effectiveness of Rainbow [9], we also use it as our target system for comparability purposes. On one side, the evaluation results demonstrate the feasibility of our reference model in

terms of the *settling time* (i.e., the time required to adapt a monitoring strategy upon changes in control objectives) of our DYNAMICO implementation. On the other side, these results demonstrate the effectiveness of using DYNAMICO to improve context-awareness in SAS solutions. Finally, based on the findings of our comparative evaluation, we analyze the suitability of Znn.com as a benchmark target system to compare self-adaptation implementations.

This paper is organized as follows. Section II introduces the dynamic SOA governance case study. Section III explains our DYNAMICO reference model. Sections IV and V evaluate the applicability of DYNAMICO and its support for changes in system requirements at runtime. Section VI presents our comparative evaluation analysis and results on the effectiveness and feasibility of DYNAMICO. Section VII discusses related work, and Section VIII concludes the paper.

II. CASE STUDY: DYNAMIC SOA GOVERNANCE BASED ON SELF-ADAPTATION

Znn.com has been proposed as a target system exemplar for the SEAMS community. This system imitates a news website that follows a three-tier architecture consisting of databases, application servers, and clients [9]. The adaptation goal of Znn.com is to provide news contents within an acceptable response time and quality. To satisfy this goal, the adaptation strategy is based on the modification of parameters of the Apache configuration file, including the content pages. Thus, Znn.com was implemented to support parametric adaptation based on *performance, cost,* and *content fidelity*. The control actions (i.e., the mechanisms that affect the target system) supported by Znn.com consist of switching the server contents mode from multimedia to text and vice versa, and incrementing or decrementing the Apache server thread pool size. The context variables required to be observed are *request-response time, server load,* and *connection bandwidth*.

Based on DYNAMICO, we implemented a dynamic SOA governance mechanism that adapts Znn.com to guarantee the satisfaction of a performance SLA. *Throughput*, defined as the time spent to process a news request $(ms/request)$, is the quality factor we defined for the initial SLA. Later, while the system is operating, the performance SLA is renegotiated by adding *capacity* as a new quality factor. This new contracted condition states that the way of delivering the contents must vary according to the connection bandwidth. That is, depending on the available bandwidth, news contents delivered to clients will be based on either text or multimedia. Our solution implements a dynamic monitoring system that deploys new context gatherers and monitors at runtime. These elements, required to satisfy new monitoring requirements, are automatically synthesized from SLAs and deployed without sensible interruptions in the execution of neither the target system nor the adaptation mechanism.

III. THE DYNAMICO REFERENCE MODEL

DYNAMICO provides guidelines for designing the software architecture of SAS systems that are affected by changes in context situations and adaptation goals [3]. In our opinion, for a software system (i.e., target system) to become effectively context-driven self-adaptive, it should incorporate at least three types of feedback loops to control three levels of dynamics: (a) the regulation of the target system requirements satisfaction; (b) the continuous accomplishment of adaptation goals and the preservation of the target system quality attributes under changing conditions of execution; and (c) the relevance of the context monitoring infrastructure according to the varying execution environment and changing adaptation goals. Figure 1 is an abstraction of our DYNAMICO reference model.

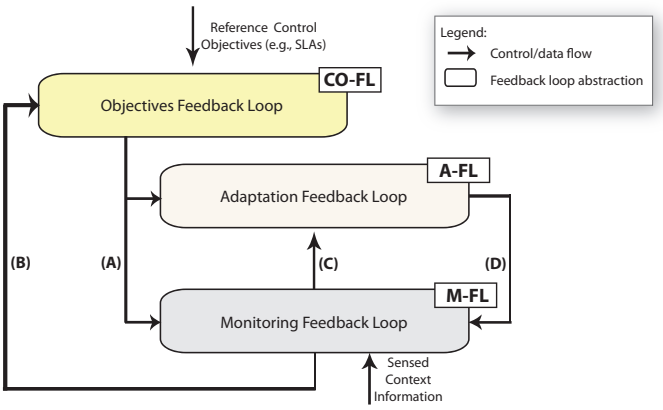

Figure 1. The three levels of dynamics in a context-driven self-adaptive software systems [3].

The separation of concerns among these three levels of dynamics made explicit by DYNAMICO is particularly crucial for cases such as the service-oriented news system presented in the previous section. In this scenario, changes in SLAs at runtime require the adaptation of not only the target system, but also the monitoring infrastructure to preserve the relevance of the adaptation mechanism with respect to the current contracted conditions. However, the automatic reconfiguration of the monitoring infrastructure is impractical having the context manager tightly coupled to the adaptation mechanism. Similarly, the explicit control of changes in SLAs (i.e., control objectives) requires separate instrumentation. Figure 2 depicts a detailed view of the feedback loops for the three levels of dynamics presented in Fig. 1.

A. The Control Objectives Feedback Loop (CO-FL)

The Control Objectives Feedback Loop (cf. CO-FL in Fig. 2) addresses the first level of dynamics specified by DYNAMICO. It governs changes in control objectives (e.g., SLAs) with the collaboration of the adaptation feedback loop (A-FL) and the monitoring feedback loop (M-FL). We define requirements and adaptation properties as system variables to be controlled. We refer to these variables as *control objectives* and *adaptation goals* interchangeably. Moreover, control objectives are subject to change by user-level (re)negotiations at runtime and therefore must be addressed in a consistent and synchronized way by the adaptation mechanism (i.e., A-FL) and the context manager (i.e., M-FL). For example, in

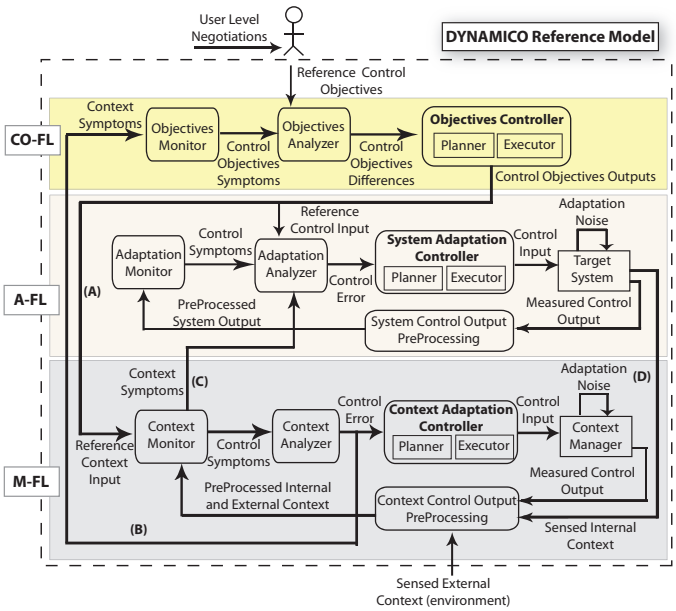

Figure 2. Our DYNAMICO reference model with a detailed view of the controllers for the three abstract levels of dynamics presented in Fig. 1 [3].

our case study, the existing performance SLA is renegotiated by adding capacity as a new quality factor. Therefore, the context monitor must keep track of two new context variables, upload and download bandwidth. Both throughput and capacity are managed explicitly as the control objectives for the adaptive system. Thus, both reference inputs, the A-FL reference control input and the M-FL reference context input, are derived automatically from control objectives and fed into the corresponding feedback loops, as illustrated by interaction (A) in Fig. 2.

B. The Adaptation Feedback Loop (A-FL)

The A-FL, the second level of dynamics, regulates the target system requirements satisfaction and the preservation of the adaptation properties. Recalling our self-adaptation scenario (cf. Sect. II), the throughput and capacity quality factors represent system requirements. Due to changing SLAs, the satisfaction of these requirements depends on the adaptive capabilities of the news system platform. For example, under the first version of the SLA, the system reconfigures itself to increase/decrease its server thread pool capacity according to the demand. After renegotiating the SLA, the capacity of the client connection becomes a new context variable to be monitored. Thus, according to the available bandwidth, the A-FL will trigger the adaptation of the system by changing the format of the delivered contents from multimedia to text. For this, the A-FL gathers symptoms from the target system through context monitors provided by the M-FL (cf. Label (C) in Fig. 2).

C. The Context Monitoring Feedback Loop (M-FL)

The M-FL in Fig. 2 represents a dynamic context manager, the third level of dynamics specified by DYNAMICO. The reference context inputs correspond to the context monitoring requirements and are derived from the CO-FL reference control objectives. In our case study the reference control objectives are defined as contracted quality of service (QoS) conditions, also called service level objectives (SLOs) in the performance SLA. Thus, the context monitoring requirements are derived from the metrics defined in this SLA. The context analyzer decides about the adaptation of the monitoring strategy. The context adaptation controller is responsible for defining and triggering the execution of the adaptation plan to adjust the context manager (i.e., the target system of the M-FL).

Referred to our case study, the addition of the new bandwidth monitoring requirements, caused by the renegotiation of the performance SLA, implies the architectural reconfiguration of the monitoring infrastructure. This reconfiguration consists of deploying two new context gathering components to sense upload and download bandwidth connections, and two new context monitors that implement the monitoring logic associated with these context variables.

IV. APPLICABILITY OF DYNAMICO

This section explains how we derived, from our DYNAMICO reference model, a self-adaptation implementation that governs the accomplishment of SLAs for the Znn.com news system. This implementation realizes the three levels of dynamics of DYNAMICO: The CO-FL to keep track of SLA negotiations, the A-FL to adapt the target system Znn.com, and the M-FL to support dynamic context monitoring. As a reference model, DYNAMICO is independent of the technologies, middleware and frameworks that can be used to realize the functionalities required by each of its three levels of dynamics. The implementation of DYNAMICO presented in this paper uses SMARTERCONTEXT [6] to realize the M-FL, and QOS-CARE [7] to support the dynamic reconfiguration of both the SMARTERCONTEXT monitoring infrastructure and the Znn.com system. We demonstrate the applicability of DYNAMICO by implementing the software architectures that we designed to realize its three levels of dynamics. The architectures presented in this section follow the service component architecture (SCA) specification [10], which provides a model for composing and executing applications based on service-oriented architecture principles. The following two subsections introduce SMARTERCONTEXT and QOS-CARE in the context of the DYNAMICO-based implementation for our case study. The remaining subsections explain the realization of the three levels of dynamics specified by the DYNAMICO reference model.

A. The SMARTERCONTEXT *Monitoring Infrastructure*

SMARTERCONTEXT, created by Villegas and Müller, is a dynamic context monitoring system developed to support changes in context management requirements at runtime [6].

In the solution described in this paper, SMARTERCONTEXT implements the M-FL defined by DYNAMICO and supports changes in context monitoring strategies dynamically through the self-reconfiguration of both the architecture of its monitoring infrastructure (i.e., structural adaptation) and the

business logic of its monitoring conditions (i.e., behavioral adaptation). These changes may imply the deployment of new context gatherers, processing and provisioning components, or the modification of existing monitoring logic. In our case study the adaptation of SMARTERCONTEXT is triggered by the (re)negotiation of SLAs. The reference inputs used by SMARTERCONTEXT correspond to Resource Description Framework (RDF) models that specify SLAs as reference control objectives with explicit monitoring requirements. Control actions correspond to arithmetic and logic expressions in the form of parameters that affect the behavior of SMARTERCONTEXT, and discrete operations that affect its structure.

B. The QoS-CARE Adaptation Mechanism

QOS-CARE (QoS Contract-Aware Reconfiguration systEm), created by Tamura *et al.* [11], [7], is the self-adaptation framework we used to implement the A-FL of DYNAMICO (i.e., to adapt Znn.com) and to support the dynamic architectural reconfiguration of the SMARTERCONTEXT infrastructure (i.e., the dynamic monitoring infrastructure). QoS-CARE provides an SCA layer for dynamic reconfiguration of component-based and service-oriented architectures that runs on top of SCA implementations. From the SCA middleware perspective, QOS-CARE is a complementary layer for SCA platforms that provides self-reconfiguring capabilities, in particular to preserve QoS contracts. The version of QOS-CARE we used for the case study presented in this paper was configured to be executed on top of the FRASCATI middleware, an open-source and multi-scale SCA implementation developed by Seinturier *et al.* [12]. QOS-CARE is rooted in extensions of formal models to guarantee its reliability as an architectural reconfiguration mechanism for service-oriented software systems. In our implementation of DYNAMICO these extensions are used to model the component-based structures of the target system and its reconfiguration rules. In particular, QOS-CARE extends typed and attributed graph transformation systems for the specification of these structures and reconfiguration rules, and finite state machines for the specification of transitions among the target system states.

C. Realizing the Control Objectives Feedback Loop (CO-FL)

Figure 3 presents the general architecture of SMARTERCONTEXT, which implements the CO-FL and M-FL of the self-adaptive solution for SOA governance in our case study. `SmarterContext` is the main composite of the architecture and includes four general components. `SmarterContextGUI` provides an abstraction of the CO-FL and allows system administrators to modify system goals (e.g., SLAs) at runtime. `MonitoringF-Loop` corresponds to the M-FL and receives from the CO-FL (i.e., `SmarterContextGUI`) the control objectives (COb) specification (e.g., the performance SLA for Znn.com) that will drive both the adaptation process and context manager. `ContextManager` includes components to update the inventory of monitoring components managed dynamically. This is for introspection purposes. `DynamicMonitoringInfrastructure` corresponds to the adaptive part of the monitoring infrastructure, and is controlled by the M-FL (cf. `MonitoringF-Loop`). This component implements the context gathering, processing and provisioning tasks, which are distributed on third party processing nodes, that is, on the computational infrastructure of context consumers and providers (e.g., on the Znn.com news system infrastructure).

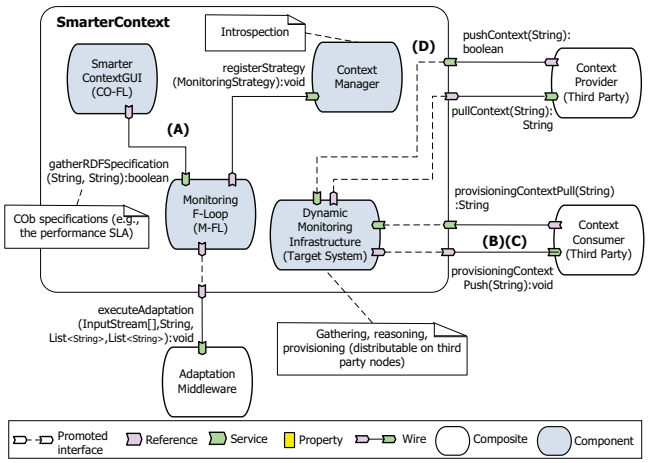

Figure 3. The CO-FL orchestrating the interactions among the three levels.

The CO-FL orchestrates the interactions among the three levels of dynamics of DYNAMICO (cf. labels (A), (B), (C) and (D) in Fig. 3). Label (A) corresponds to the interaction between the CO-FL and M-FL that enables the SMARTERCONTEXT engine of the M-FL to receive the specifications of control objectives and their changes at runtime (e.g., SLAs and their modifications). Labels (B) and (C) represent the interactions that allow SMARTERCONTEXT to notify context monitoring facts to the system objectives manager (e.g., a user) and the adaptation mechanism (e.g., Znn.com). Both the system objectives manager and adaptation mechanism are abstracted as context consumers in this architecture (cf. composite `ContextConsumer` in Fig. 3). Label (D) represents the interaction through which the monitoring infrastructure (i.e., the target system controlled by the M-FL) gathers context information about the situation of the adaptive system, which acts as a context provider (cf. composite `ContextProvider`). Composite `AdaptationMiddleware` represents the QOS-CARE/FRASCATI adaptation middleware and is invoked by `MonitoringF-Loop` to trigger the adaptation of the SMARTERCONTEXT architecture.

D. Realizing the Monitoring Feedback Loop (M-FL)

In our case study, the implementation of the context monitoring feedback loop enables our monitoring infrastructure with dynamic capabilities to adapt (i) the context monitoring logic that evaluates gathered context against contracted conditions, and (ii) the architecture of the context gathering and provisioning infrastructure. Figure 4 illustrates our realization of the M-FL as the `MonitoringFeedbackLoop` composite, highlighted with Label ①, which is the concrete realization

of the MonitoringF-Loop component presented in Fig. 3. Label ② highlights the MonitoringInfrastructure composite, which contains the adaptive monitoring infrastructure (i.e., the "target system" of the M-FL). Finally, Label ③ highlights the AdaptationMiddleware composite, which contains the QoS-CARE components in charge of adapting the monitoring infrastructure.

The first component of M-FL is ContextMFLMonitor (cf. Fig. 4). This component receives the COb specification in XML/RDF format from the CO-FL, creates an RDFSpecification object from the received specification, looks for a previous version of this COb specification, stores the new version in its knowledge base, and provides the component ContextMFLAnalyzer with two RDFSpecification objects that represent the new and former versions of the COb specification (e.g., when the performance SLA of Znn.com is renegotiated by adding the capacity SLO).

The second component of M-FL is ContextMFLAnalyzer, which analyzes changes between the new and former versions of COb specifications, and specifies these changes in an RDF model. After analyzing these changes, this component invokes ContextMFLPlanner and provides it with the new version of the COb specification and the model that specifies the changes.

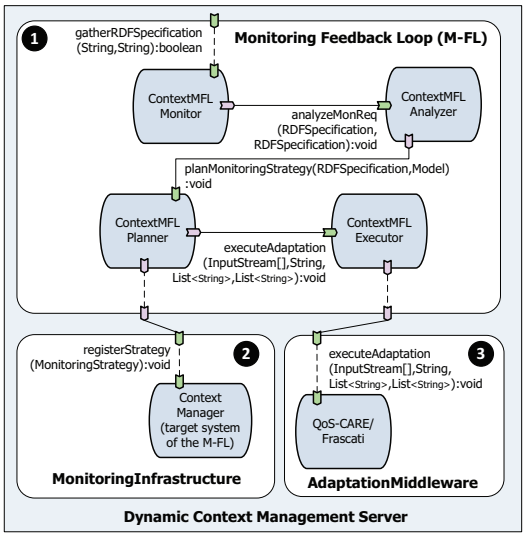

Figure 4. Realization of the M-FL

The third component of M-FL is ContextMFLPlanner. This component synthesizes new monitoring strategies as well as changes in existing ones. We define monitoring strategies as an object that contains a set of implementation files (i.e., .class files) for the SCA components to be deployed at runtime, the specification of the corresponding SCA composite (i.e., an XML file), and two lists that specify SCA services and corresponding references. These services and references allow the connection of third-party sensors and gatherers to the endpoints of the SMARTERCONTEXT infrastructure.

The last component of composite M-FL is ContextMFLExecutor, which invokes the adaptation services of the AdaptationMiddleware composite. The QoS-CARE components of this composite adapt the context monitoring infrastructure by deploying the new context gatherers and monitors, and binding their services to the references of the sensors deployed on the target system infrastructure. To perform this adaptation, the QoS-CARE instance of the M-FL invokes the corresponding service of the QoS-CARE instance of the A-FL.

E. Realizing the Adaptation Feedback Loop (A-FL)

Figure 5 presents our realization of the A-FL for our case study. In this realization, composite DynamicMonitoringInfrastructure (cf. Label ②) contains the components that are deployed dynamically by the M-FL to monitor the relevant context according to the contracted conditions of the SLA. For instance, within this composite, the highlighted ContextGatheringAndPreprocessing and ContextProcessing components are deployed dynamically to monitor one of the new context variables (i.e., download bandwidth) that became relevant after renegotiating the performance SLA in our case study. Components ContextProcessingThroughput, ContextProcessingBandwidthDown and ContextProcessingBandwidthUp (the latter not depicted in the figure) not only notify the composite AdaptationMiddleware (cf. Label ①) about the need for adaptation but also system administrators. In the AdaptationMiddleware composite, the QoS-CARE instance implements the adaptation analyzer and adaptation controller (i.e., planner and executor) of the A-FL. The executor adapts the Znn.com target system (cf. composite TargetSystem, Label ③), by modifying the Apache configuration file or changing the news contents delivery format.

V. DYNAMIC CHANGES IN CONTROL OBJECTIVES

For the CO-FL to orchestrate the interactions among the three levels of dynamics, it requires runtime models that provide an explicit mapping between control objectives (e.g., SLAs) and both monitoring requirements and conditions that trigger self-adaptation. In our implementation, these models correspond to RDF graphs, named *control objectives (COb) specifications*. From these, SMARTERCONTEXT (i) synthesizes monitoring strategies for implementing the context monitoring mechanism required to support the adaptation process, and (ii) identifies changes in existing monitoring strategies when existing system control objectives change or new ones appear. In our case study COb specifications correspond to SLAs.

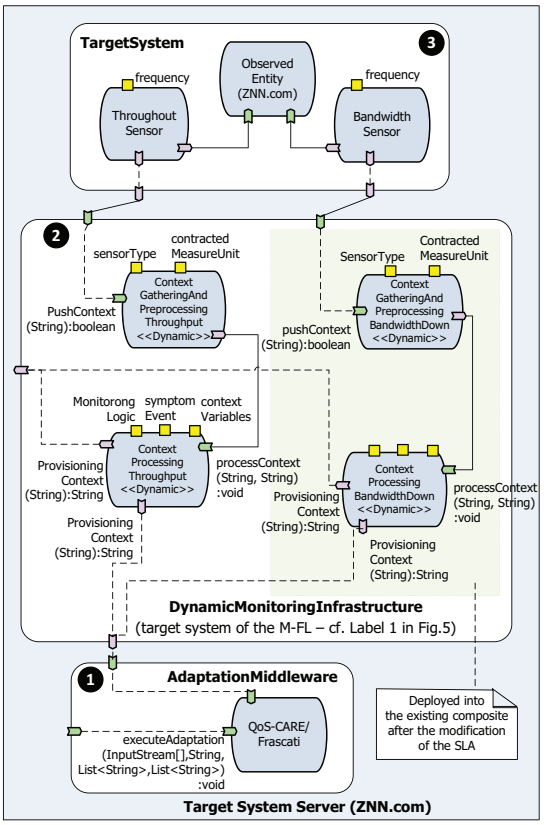

Figure 5. Realization of the A-FL

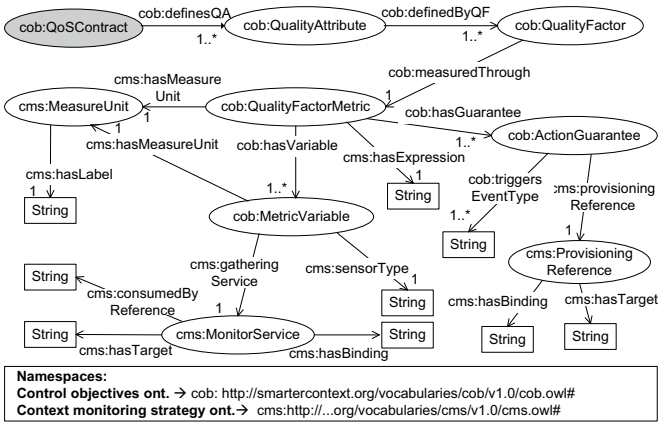

Figure 6. The control objectives (COb) ontology

A. Control Objectives Specifications

Figure 6 depicts the *control objectives (COb)* ontology for QoS contracts in SMARTERCONTEXT.[1] This ontology allows the specification of control objectives (e.g., SLAs in the case of Znn.com) mapped to elements of both monitoring strategies and adaptation mechanisms represented by entities derived from our *context monitoring strategy (cms)* ontology.[2] Even though cardinalities are not usually specified in RDF graph-based representations, Fig. 6 represents them for explanation purposes.

The highlighted node, `cob:QoSContract`, represents the root type of quality-driven COb specifications. Our ontology for the specification of QoS contracts, particularly SLAs in SOA environments, is based on the characterization of SLAs contributed by SEI researchers [13]. According to them, a properly specified SLA must include (i) the metrics to be collected, (ii) the entity that will collect these metrics and how, and (iii) the actions to be taken when the service does not meet the contracted conditions. In the COb specifications used in our case study, these aspects correspond to (i) quality factor metrics, (ii) monitoring services, and (iii) action guarantees associated with adaptation events or notifications to system administrators. Quality-driven COb specifications must include at least one quality attribute. Quality attributes are instances of type `cob:QualityAttribute` and represent measurable qualities that can be observed automatically through a sensing interface (e.g., performance). Each quality attribute is composed of at least one quality factor (e.g., throughput). Quality factors are instances of type `cob:QualityFactor` and correspond to quality properties that allow the measurement of quality attributes. Each quality factor must be associated with at least one metric. Metrics are instances of type `cob:QualityFactorMetric` and define the variables (`cob:MetricVarible`), the evaluation expression (`String`), the measure unit (`cms:MeasureUnit`), and the action guarantees (`cob:ActionGuarantee`) required to control the preservation of the contracted quality attributes. A metric must be associated with one and only one evaluation expression, and one unit of measurement. Action guarantees are instances of type `cob:ActionGuarantee` and specify the event types (`String`), with corresponding endpoints (`cms:ProvisioningReference`), to be triggered when the contracted conditions are violated (e.g., to adapt the Znn.com system).

The mapping between control objectives and monitoring requirements is realized by associating elements of type `cob:MetricVariable` with instances of type `cms:MonitorService`. These instances provide the mechanisms to collect the metrics required to assess the contracted qualities. Each metric is associated with at least one metric variable, and a metric variable with one and only one sensor type and one monitoring service. The type `cms:MonitorService` allows the specification of the URI of the reference that will consume the monitoring interface when the gathering is based on a pushing mechanism (i.e., the source sends the sensed data to the SMARTERCONTEXT infrastructure). Similarly, `cms:MonitorService` allows the specification of the URI of the target reference or binding when the gathering mechanism is realized through a pulling mechanism (i.e., the SMARTERCONTEXT infrastructure pulls the sensed data from the source).

The mapping between control objectives and self-adaptation conditions is realized through the specification of action

[1]http://smartercontext.org/vocabularies/cob/v1.0/cob.owl
[2]http://smartercontext.org/vocabularies/cms/v1.0/cms.owl

guarantees. Each object of type `cob:ActionGuarantee` is associated with an event type that corresponds to the symptom used by the analyzer of the A-FL to decide about adaptation actions to be taken. Action guarantees also specify an object of type `cms:ProvisioningReference` that defines the binding or target URI to be consumed. Besides adaptation actions, action guarantees can imply notifications to system administrators.

Figure 7 partially represents a CObs specification for the performance SLA that resulted from the first negotiation in our case study.[3] Namespace `qa:` corresponds to the vocabulary that characterizes quality attributes mapped to quality factors. This version of the performance SLA defines a throughput quality factor, measured through a throughput metric (`qa:ThroughputMetric`) that is composed of a single variable (`qa:processingTime`). This variable is involved in the metric expression $?processingTime \leq 2000$, measured in terms of *ms/request* (as defined by the element `qa:ThroughputMeasureUnit`) and associated with a `cms:MonitorServiceType` identified as `sla.rdf#gatheringServiceThroughput`. The action guarantee defined for the throughput metric (`sla.rdf#ActionGuaranteeThroughput`) is associated with two provisioning references. The first one, `sla.rdf#adaptTargetSystem` is to invoke the service in charge of activating the adaptation process. The second one, `sla.rdf#notifyAdministrator`, is to inform business administrators about the violation of the contracted throughput conditions.

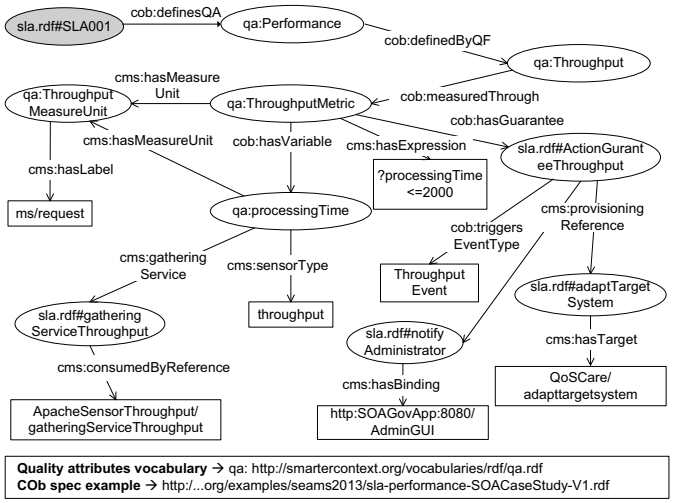

Figure 7. A CObs specification example for the throughput quality attribute defined in the first negotiation of the performance SLA in our case study.

B. Synthesizing Monitoring Strategies at Runtime

SMARTERCONTEXT synthesizes and implements monitoring strategies dynamically from CObs specifications such as the one depicted in Fig. 7. A monitoring strategy is implemented as a `DynamicMonitoringInfrastructure` composite (cf. Fig. 5) that specifies components for context gathering, pre-processing, monitoring, and provisioning. These components include their corresponding service interfaces, references, and properties. In particular, each `cob:QualityFactorMetric` element generates a `ContextProcessing` component whose monitoring logic corresponds to the value of the `cms:hasExpression` property; each `cob:MetricVariable` produces a `ContextGatheringAndPreprocessing` component that gathers one and only one context value from the corresponding sensor type (i.e., the value of the `cms:sensorType` property) deployed at the context provider party; and each `cob:ActionGuarantee` element generates a `ContextProvisioning` component whose `symptomEvent` property corresponds to the value of the `cob:triggersEventType` of the CObs specification.

Upon the renegotiation of SLAs, the M-FL analyzer calculates changes in monitoring requirements by comparing a new CObs specification (e.g., the renegotiated performance SLA[4]) with its previous version. Then, the planner element of the M-FL generates the adaptation plan that will modify the monitoring infrastructure by deploying new context gathering, processing and provisioning components, or modifying existing monitoring logic. If the CObs specification has no previous version (e.g., it is a new SLA), the planner generates the new strategy from scratch. Finally, the M-FL executor performs the adaptation of the monitoring infrastructure.

VI. EVALUATION

In this section we assess and compare the implementations of CMU's Rainbow and our DYNAMICO reference model. As mentioned above, Cheng *et. al* introduced the Znn.com system as a candidate target system exemplar to benchmark self-adaptation mechanisms comparatively to evaluate the effectiveness of Rainbow. To be able to compare Rainbow and DYNAMICO, we use the Znn.com exemplar as our target system and apply the same criteria introduced by Cheng *et. al* in [8] and [9]. However, it is worth noting that in [8] Rainbow was not applied to Znn.com, but to a videoconference target system.

A. Evaluation Criteria

Cheng *et. al* used the following three factors for evaluating the effectiveness of Rainbow to adapt Znn.com: (i) performance, measured in terms of the time required to perform the adaptation (i.e., called settling time in control theory [14]), and the runtime overhead caused by executing Rainbow with Znn.com as the adaptation mechanism; (ii) the engineering effort required to use Rainbow to add self-adaptive capabilities to Znn.com; and (iii) the capability of Rainbow to maintain

[3]http://smartercontext.org/examples/seams2013/sla-performance-SOACaseStudy-V1.rdf

[4]http://smartercontext.org/examples/seams2013/sla-performance-SOACaseStudy-V2.rdf

system quality attributes under changing conditions of execution. The following sections present the results of our case study evaluation using these factors.

B. Performance

To evaluate the performance of our DYNAMICO implementation we realized the adaptation scenario described in Sect. II with several platform configurations. The different hardware and software configurations were based on Intel i3@2.4Ghz processors with 4 GB of RAM running GNU/Linux Fedora 16 with non-relevant services and applications shut down. For the context monitoring infrastructure we used SMARTERCONTEXT v1.5,[5] whereas for the SCA platform with autonomous reconfiguration capabilities we used QOS-CARE v1.3[6] and FRASCATI v1.4[7] with Java 1.6.0_23 with 128 MB of RAM.

We measured the performance of the DYNAMICO implementation in terms of (i) the settling time of both the adaptation of the target system and the adaptation of the monitoring infrastructure, and (ii) the overhead caused by adding our DYNAMICO implementation in the execution of Znn.com. To obtain these measurements we configured an environment as an SCA architecture (except for Znn.com, which is a PHP application) with the following four servers:

1) Target system, adaptation infrastructure (A-FL) and monitoring infrastructure of the M-FL: this server executes Znn.com with Apache, its execution platform, the SMARTERCONTEXT gathering and processing components, and QOS-CARE/FRASCATI as the monitoring infrastructure's and target system's adaptation mechanism (cf. Fig. 5). To enable the monitoring of the variables of interest on Znn.com (i.e., throughput in terms of ms/request and bandwidth in terms bytes transfer rate), we developed the required sensors and corresponding interfaces.

2) Control objectives management (CO-FL): executes a GUI to change the control objectives, that is, the SLAs to be satisfied by the target system according to the scenario specified in our case study.

3) Dynamic context management (M-FL): executes SMARTERCONTEXT and QOS-CARE/FRASCATI as the adaptation mechanism of the monitoring infrastructure (cf. Fig. 4).

4) Client testers: to automate the execution of our experiments and measure settling times as well as overhead we designed two client testers to be executed in multiple machines to (i) modify the control objectives of the target system (i.e., to simulate the renegotiation of the SLA and thus force the adaptation of the monitoring infrastructure), and (ii) change the execution conditions of Znn.com (i.e., to force its adaptation).

On the one hand, from the execution of the client tester that changes the SLA in the CO-FL, we obtained the settling time and overhead measurements for the adaptation of the monitoring infrastructure (cf. Table I). The first row in this table indicates that the CO-FL, which keeps track of changes in control objectives and sends COb specifications to the M-FL, takes 698 ms for the first scenario (i.e., SLA-v1 requiring one throughput gatherer and one throughput processor) and 732 ms for the second (i.e., SLA-v2 requiring the addition of two bandwidth gatherers and two bandwidth monitors—for monitoring upload and download capacity). The second row depicts the timings for the M-FL execution, which analyzes and synthesizes the adaptation plan of the monitoring infrastructure. The third row indicates the timings for instrumenting the synthesized monitoring infrastructure (i.e., deploying and binding the context processors and gatherers). The total settling times, and the measured overhead of DYNAMICO, which is practically negligible in both scenarios, represent an acceptable cost for having these two additional feedback loops for adapting the monitoring infrastructure.

Table I
AVERAGE SETTLING TIME AND OVERHEAD (MS)

	SLA-v1[ms]	SLA-v2[ms]
CO-FL	698	732
M-FL	21	29
Adaptation Instrumentation	1,131	1,579
Total settling time	1,850	2,340
Overhead	3	3

It is worth noting that Rainbow has no support for adapting the monitoring infrastructure of the adaptation mechanism. Thus, Rainbow and DYNAMICO cannot be compared in this regard. However, throughout this paper we have demonstrated that the CO-FL and M-FL, the two additional feedback loops specified by DYNAMICO, improve the context awareness of our adaptation mechanism.

On the other hand, from the execution of the client tester that changes the Znn.com execution conditions to induce its adaptation, we obtained an average settling time of 84 ms and an overhead of 2 ms. Nonetheless, the comparative analysis of this settling time presented several difficulties. First, it makes no sense to disaggregate this measurement, given that we used the same strategy as Rainbow to adapt Znn.com. That is, by modifying parameters of the configuration file of Apache, for example to vary the number of server thread pool. Even though this kind of adaptation is argued to be architectural, because it makes Apache to configure internally a different number of server threads (i.e., software components in its internal architecture), it can be argued that it is parametric. This is because the adaptation mechanism modifies a parameter (a single value) that affects the target system behavior, not its actual software structure. Of course, as Znn.com is a PHP monolithic application without introspection capabilities, modifying its software structure is impractical. Second, the settling time of Rainbow to perform a target system adaptation was measured for a videoconference system in a different

[5]Available from http://gforge.icesi.edu.co/svn/smartercontext
[6]Available from https://scm.gforge.inria.fr/svn/scesame/qos-care
[7]Available from svn://svn.forge.objectweb.org/svnroot/frascati

paper (i.e., [8]) than the one that used Znn.com. In [8], the reported measurement on the adaptation settling time is 2,100 ms and 2,700 ms for two different scenarios. Despite of the aforementioned difficulties to perform a comparative analysis, the obtained results demonstrate the feasibility of implementing our reference model, considering the complexity of combining three different types of feedback loops in the adaptation mechanism.

C. Engineering Effort

Measuring the engineering effort to develop a software system, on an absolute scale and independently of human factors, is well known to be a challenging task. This often means that engineering efforts of different development teams cannot be compared. To evaluate the effort of enabling DYNAMICO to adapt Znn.com with that of Rainbow, we considered the tasks to be developed in similar conditions. These tasks are (i) development and testing of three types of required sensors (12 h in DYNAMICO vs. 49 h in Rainbow), (ii) development and testing of adaptation scripts (8h vs. 21h), and (iii) architecture (target system-adaptation mechanism) deployment configuration (11 h vs. 24 h), for a total of 31 h vs. 94 h.[8] The ratio of the development effort DYNAMICO/Rainbow is around 1/3 and is observed in all of the evaluated factors, approximately. Given the available information, we found two possible explanations for this difference. The first is the degree of separation of concerns between the target system and the adaptation mechanism. DYNAMICO provides our implementation not only with a well defined structure of subsystems, each addressing different levels of dynamics, but also with the characterization of the interactions among them. Having clear these two aspects is fundamental to maintain separated the addressed concerns and functionalities of each subsystem. As a consequence, the visibility of the different feedback loops is maintained from architecture design to code, which favors the maintainability and reusability of our adaptation mechanism. The second is the chosen model for the system architecture, which is SCA in our DYNAMICO implementation, and ACME in Rainbow. Our implementation uses SCA interfaces as the communication means among all of the system components, which helps to maintain well defined limits among the subsystems. In any case, the obtained measurements show that the effort required to tailor our DYNAMICO implementation to adapt Znn.com is significantly lower than tailoring Rainbow for the same goal, and definitely lower than developing self-adaptive capabilities in Znn.com from scratch.

D. Maintaining System Quality Attributes

Znn.com is a small PHP monolithic application, executed in Apache. This fact has an unfortunate implication for the evaluation of the capability of any adaptation mechanism to maintain Znn.com's quality attributes. Indeed, the possibilities for adapting Znn.com/Apache are restricted to the strategy implemented by Rainbow, that is, by changing some parameters of the Apache configuration file and restarting it. Therefore, any adaptation mechanism required to maintain the same quality attributes maintained by Rainbow in Znn.com would necessarily use this strategy. As a consequence, and given that our DYNAMICO implementation adapts Znn.com using exactly the described strategy, it would obtain the same evaluation than Rainbow regarding this factor. Nonetheless, by virtue of the improved context awareness obtained by the two extra feedback loops specified in DYNAMICO, CO-FL and M-FL, our adaptation mechanism is able to maintain quality attributes that depend on context information not sensed by the current monitoring infrastructure. Thus, the reliability of the decision-making process for the preservation of quality attributes is also improved.

[8]The Rainbow adaptation effort data are based on [8].

VII. Related Work and Discussion

Raising the visibility of feedback loops is crucial for the engineering of adaptation mechanisms for systems that must cope with changing requirements dynamically. This is especially important for managing changes in adaptation goals, and based on these changes, adjusting monitoring strategies with the goal of preserving context-awareness. Other reference models have been proposed to contribute to the engineering of SAS systems. In particular, ACRA [15] and Kramer and Magee's [16], which follow multi-layer architectures that expose multiple feedback loops. Nevertheless, even though these models specify several feedback loops, in general they assume closed and controlled context environments where system and monitoring requirements are specified at design time, remaining immutable at runtime. In DYNAMICO the separation of concerns goes beyond the decoupling of adaptation mechanisms from managed systems, and the hierarchical organization of their interactions. Distinguishing features of DYNAMICO are its CO-FL that, according to changes in adaptation goals, governs *both* the A-FL and the M-FL; and its M-FL that, through its interactions with the CO-FL and the A-FL, improves not only context-awareness but also the accomplishment of changing adaptation goals (e.g., quality attributes). Similarly to other reference models for SAS systems, DYNAMICO relies on the MAPE-K loop to characterize the components that define control objectives, adaptation and dynamic monitoring feedback loops. However, DYNAMICO is independent of the particular strategies and technologies used for implementing self-adaptation [3].

Concerning dynamic monitoring to support self-adaptation under changing adaptation goals, previously we analyzed an important number of representative SAS systems and found that none of them supports dynamic monitoring under changing control objectives [1]. Moreover, many of these approaches simulate their monitoring mechanisms. In contrast, the DYNAMICO implementation presented in this paper is a self-adaptation solution where the monitoring infrastructure adapts itself to address changes in monitoring requirements. With this, we guarantee that the adaptation mechanism is fed with symptoms relevant to the actual adaptation goals, which also change continuously. We conducted several experiments to compare our DYNAMICO implementation with Rainbow.

From these experiments we obtained useful results. First, we were able to confirm that implementations based on our DYNAMICO reference model are not only viable, but also can improve the context-awareness of self-adaptation. Second, we found that the SEAMS community still lacks suitable target system exemplars and tools for benchmarking self-adaptation solutions. Znn.com is a good baseline but it is not versatile enough in supporting different adaptation strategies. Based on the Rainbow/Znn.com's publicly available implementation and documentation, we conclude that Znn.com could be further developed to support the validation of structural and behavioral adaptation mechanisms better. SAS systems research requires exemplars that support the wide range of control actions (i.e., adaptation effectors) and sensors required by self-adaptation approaches, and that implement standard interfaces to interact with them. For example, using the current version of Znn.com it is impractical to assess the adaptation effectiveness of approaches that rely on architectural reconfiguration of component-based or service-oriented systems. Moreover, adaptation mechanisms to be evaluated by adapting Znn.com must be based on parametric changes applied to the properties of the Apache server configuration file (i.e., the server thread pool size), or application variables such as the delivery format of news contents (i.e., text and multimedia). To advance in the evaluation of SAS systems, we should extend Znn.com and propose new benchmark target systems. Equally important for the assessment of SAS systems is for the SEAMS community to agree on a set of standardized properties [1].

VIII. CONCLUSIONS

In this paper we demonstrated the effectiveness and feasibility of our DYNAMICO reference model through the evaluation of its implementation and application in an experimental case study. This evaluation showed that DYNAMICO is especially effective for building context-aware self-adaptation mechanisms where the monitoring infrastructure must be self-adaptive to address changing requirements. To realize dynamic context monitoring in our DYNAMICO implementation (i.e., with the CO-FL and M-FL acting in coordination), we used the SMARTERCONTEXT monitoring infrastructure with the QOS-CARE/FRASCATI middleware as its adaptation mechanism. To compare our DYNAMICO implementation with that of the Rainbow/Znn.com system, we used the Znn.com exemplar as target system. This allowed us to analyze the suitability of Znn.com as a benchmark system for self-adaptation approaches. Our experimental results demonstrate, in terms of performance, the feasibility of instrumenting adaptation mechanisms with the two extra levels of dynamics of DYNAMICO. This is to support dynamic monitoring according to changes in requirements and thus favoring context awareness. We also analyzed the engineering effort of building DYNAMICO-based implementations, and their suitability to preserve system quality attributes. The obtained results support the effectiveness of our reference model for engineering context-aware SAS systems.

ACKNOWLEDGMENTS

This work was funded in part by University of Victoria (Canada), the National Sciences and Engineering Research Council (NSERC) of Canada, IBM Corporation, Icesi University (Colombia), and Ministry of Higher Education and Research of Nord-Pas de Calais Regional Council and FEDER under Contrat de Projets Etat Region (CPER) 2007-2013.

REFERENCES

[1] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A Framework for Evaluating Quality-driven Self-Adaptive Software Systems," in *Proceedings 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*. New York, NY, USA: ACM, 2011, pp. 80–89.

[2] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cikic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A second Research Roadmap*. Springer, 2013, vol. 7475, pp. 1–32.

[3] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, ser. LNCS. Springer, 2013, vol. 7475, pp. 265–293.

[4] H. Müller, M. Pezzè, and M. Shaw, "Visibility of Control in Adaptive Systems," in *Proceedings 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008)*, 2008, pp. 23–26.

[5] Y. Brun, G. D. M. Serugendo, C. Gacek, H. M. Giese, H. Kienle, M. Litoiu, H. A. Müller, M. Pezzè, and M. Shaw, *Engineering Self-Adaptive Systems through Feedback Loops*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2009, vol. 5525, pp. 48–70.

[6] N. M. Villegas, "Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems," Ph.D. dissertation, University of Victoria, Canada, February 2013.

[7] G. Tamura, "QoS-CARE: A Reliable System for Preserving QoS Contracts through Dynamic Reconfiguration," Ph.D. dissertation, University of Lille 1 - Science and Technology, and Universidad de Los Andes, 2012.

[8] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," *IEEE Computer*, vol. 37, pp. 46–54, 2004.

[9] S.-W. Cheng, D. Garlan, and B. Schmerl, "Evaluating the Effectiveness of the Rainbow Self-Adaptive System," in *Proceedings 2009 Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009) ICSE Workshop*, may 2009, pp. 132–141.

[10] OSOA, "SCA Assembly Model version 1.0," http://www.osoa.org, 2007.

[11] G. Tamura, R. Casallas, A. Cleve, and L. Duchien, "QoS Contract-Aware Reconfiguration of Component Architectures Using E-Graphs," in *Formal Aspects of Component Software*, ser. LNCS, vol. 6921. Springer, 2012, pp. 34–52.

[12] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A Component-based Middleware Platform for Reconfigurable Service-Oriented Architectures," *Software: Practice and Experience*, vol. 42, no. 5, pp. 559–583, 2012.

[13] P. Bianco, G. Lewis, and P. Merson, "Service Level Agreements in Service-Oriented Architecture Environments," Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-2008-TN-021, 2008. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/08tn021.cfm

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[15] IBM Corporation, "An Architectural Blueprint for Autonomic Computing," IBM Corporation, Tech. Rep., 2006. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[16] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *Proceedings 2007 workshop on the Future of Software Engineering (FOSE 2007)*. IEEE Computer Society, 2007, pp. 259–268.